KOMUNIKACIJA PORUKAMA

REDOVI PORUKA

Poruka je mala količina podataka (na primjer, do nekoliko stotina bajtova) koja može biti poslana u red poruka. Poruci se može pridijeliti tip po kojem se može prepoznati. Svaki proces s odgovarajućom dozvolom može primiti poruku iz reda.

Red poruka može poslužiti kao semafor: stavljanje poruke u red je ekvivalentno otvaranju semafora, a uzimanje poruke iz reda ekvivalentno je zatvaranju semafora. Poziv za uzimanje poruke se normalno zablokira ako je red prazan što odgovara stanju kada je semafor na nuli.

Sustavski pozivi za rad s redovima poruka

Rad sa redovima poruka je ovdje opisan nešto detaljnije nego što su bili opisani pozivi za rad sa zajedničkom memorijom i skupovima semafora. Ipak, više detalja se može naći sa: `man msgget`, `man msgop` i `man msgctl`. Podaci potrebni za rad sa redovima poruka definirane su u datotekama `<sys/types.h>`, `<sys/ipc.h>` i `<sys/msg.h>` koje treba uključiti na početku programa. Opis bitnih struktura podataka može se naći sa `man intro`.

```
int msgget(key_t key, int flags);
```

Sustavski poziv *msgget* stvara red poruka, ili vraća identifikator reda poruka ako red već postoji. Poziv je analogan sustavskom pozivu *open*. Kao parametar prima ključ *key* i vraća identifikator reda, odnosno -1 ako dođe do greške.

Identifikator reda je vrlo sličan opisniku datoteke, osim što ga može koristiti bilo koji proces koji poznaje taj broj. Ako je postavljen bit `IPC_CREATE` u *flags*, red se kreira ako već ne postoji, a devet najnižih bitova su dozvole za korištenje reda. Dozvola za pisanje dopušta da poruka bude poslana, a dozvola za čitanje dopušta primanje poruke. Ako `IPC_CREATE` nije postavljen onda red mora postojati i u tom slučaju ova funkcija samo pronalazi identifikator reda. (Ako se za *key* stavi `IPC_PRIVATE` onda se kreira novi red bez obzira na `IPC_CREAT`.)

Dozvole pristupa u *flags* su definirane na slijedeći način:

```
00400Receive message by user
00200Send message by user
00040Receive message by group
00020Send message by group
00004Receive message by others
00002Send message by others

struct msgbuf {

      long mtype;
      char mtext[1];

};

int msgsnd(int msqid, struct msgbuf *msgp, int msgsz, int
msgflg);
```

Sustavski poziv *msgsnd* šalje poruku u red čiji je ID *msqid* dobiven primjerice od *msgget*. *msgp* pokazuje na strukturu u kojoj na prvom mjestu mora biti dugačak cijeli broj veći od nule - vrsta poruke. Ostatak te strukture ovisi o podacima koji se šalju. Interno se ostatak poruke prihvaća kao niz znakova (bajtova) duljine *msgsz*. Tip poruke omogućava primaocu da odabere iz reda poruke koje želi izvaditi, odnosno može čekati određeni tip poruke. *msgflg* je obično 0, što uzrokuje da se *msgsnd* zablokira dok je red pun. Druga mogućnost je `IPC_NOWAIT` što uzrokuje da poziv *msgsnd* vrati grešku ako je red pun. *msgsnd* vraća 0 ako uspije ili -1 ako dođe do greške.

```
int msgrcv(int msqid, struct msgbuf *msgp, int msgsz, long msgtyp,
int msgflg);
```

Sustavski poziv *msgrcv* poziva primalac poruke. *msgsz* mora biti veličina najveće poruke koja može stati u prostor na koji pokazuje *msgp.* Obzirom da primljena poruka može biti manja od toga, ovaj poziv vraća veličinu poruke ili -1 ako dođe do greške. Ako primalac želi određenu vrstu poruke onda je stavi u *msgtyp*. Inače se stavi 0 čime se uzima najstarija poruka iz reda (bez obzira na vrstu poruke). Ako je red prazan ili u njemu nema poruka tražene vrste onda će se *msgrcv* zablokirati, osim ako je *msgflg* (*msgflg* je obično 0) `IPC_NOWAIT` u kojem slučaju će se odmah vratiti -1 (greška).

```
struct ipc_perm {

      ushort cuid;/* creator user id*/
      ushort cgid;/* creator group id*/
      ushort uid;/* user id*/
      ushort gid;/* group id*/
      ushort mode;/* r/w permission*/
      ushort seq;/* slot usage sequence #*/
      key_t key;/* key*/

} ;

struct msg {

      struct msg *msg_next;/* ptr to next message on queue */
      long msg_type;/* message type */
      short msg_ts;/* message text size */
      short msg_spot;/* message text map address */

};

struct msqid_ds {

      struct ipc_perm msg_perm;/* message operation
      permissions*/
      struct msg *msg_first; /* ptr to the first message on
      the queue*/
      struct msg *msg_last; /* ptr to the last message on the
      queue*/
      ushort msg_cbytes; /* current number of bytes on the
      queue*/
      ushort msg_qnum; /* nr of messages currently on the
      queue*/
      ushort msg_qbytes; /* max nr of bytes allowed on the
      queue*/
```

```
        ushort msg_lspid; /* last process that performed
        msgsnd*/
        ushort msg_lrpid; /* last process that performed msgrcv
        */
        time_t msg_stime; /* time of the last msgsnd
        operation*/
        time_t msg_rtime; /* time of the last msgrcv
        operation*/
        time_t msg_ctime; /* time of the last msgctl
        operation*/

} ;
```

Red poruka se nakon uporabe treba obrisati. Npr. pozivom `msgctl(msqid,IPC_RMID,NULL)`. Sustavski poziv

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

obavlja jednu od tri operacije u ovisnosti o *cmd*:

`IPC_STAT` popunjava strukturu *buf* vrijednostima za red poruka *msqid*.

`IPC_SET` mijenja *msg_perm.uid*, *msg_perm.gid*, *msg_perm.mode* i *msg_qbytes* za red poruka *msqid* sa vrijednostima iz *buf*.

`IPC_RMID` uništava red poruka *msqid* i bilo koji zablokirani poziv vraća grešku u tom slučaju.

OKOLINA (*environment*)

Okolina je niz znakovnih nizova oblika "*ime=vrijednost*" koji se predaje svakom programu prilikom pokretanja. *ime* je varijabla okoline. Uobičajeno je za imena tih varijabli upotrebljavati velika slova iako to nije obavezno.

Varijable okoline se najčešće postavljaju korištenjem korisničke ljuske (vidi: `man sh`, `man csh`). U ljusci *sh* se sa:

```
ime=vrijednost
```

postavlja varijabla za samo ljusku. Tek ako se izvede:

```
export ime
```

ista varijabla se uključuje i u okolinu koju ljuska predaje programima koje poziva. `export` daje popis svih varijabli koje se predaju kao okolina programima, dok `set` daje popis svih varijabli koje poznaje sama ljuska. `unset ime` poništava definiciju varijable *ime*.

U ljusci *csh* se varijabla okoline programa definira sa:

```
setenv ime vrijednost
```

Varijable same ljuske se postavljaju sa:

```
set ime=vrijednost
```

Neke od najčešće korištenih varijabli: *logname*, *home*, *path*, *user* i *term* automatski se uključuju i u okolinu nakon ove naredbe, pa za njih nije potrebno upotrebljavati *setenv*. Prilikom uključenja u okolinu, imena ovih varijabli se pišu velikim slovima. `setenv` izlistava okolinu koja se predaje programima, dok `set` daje popis svih varijabli koje poznaje sama ljuska.

Kako se poziva *main*

Prototip prema kojem se poziva funkcija *main* svakog programa u UNIX-u je:

```
int main(int argc, char *argv[], char *envp[]);
```

argc je broj argumenata navedenih kod poziva programa, a *argv* je niz od *argc* kazaljki na te argumente kao nizove znakova. Prvi od tih nizova je ime samog pozvanog programa. *envp* je niz kazaljki na nizove znakova oblika "*ime=vrijednost*" koji čine okolinu. Posljednja kazaljka je `NULL`. Okolini se može pristupiti i na praktičniji način nego korištenjem *envp*. Zbog toga se *main* može definirati i kao:

```
int main(int argc, char *argv[]) { ... }
```

Također, program koji ne koristi nikakve ulazne parametre može definirati *main* kao:

```
int main(void) { ... }
```

main treba vratiti cjelobrojnu vrijednost jer poziv možemo pojednostavljeno zamisliti kao:

```
exit(main(argc, argv, envp));
```

Ako program završava pozivom *exit* na nekom mjestu, onda ne dolazi do povratka iz *main*. Međutim, ako *main* normalno završava, onda je potrebna povratna vrijednost koja će postati argument poziva *exit*.

Pristup varijablama okoline iz programa

envp nije pogodan za pristup varijablama okoline jer je poznat samo unutar main, a ne i u ostalim funkcijama programa. Zato postoji globalna varijabla:

```
extern char *environ[];
```

koja je, također, niz kazaljki na nizove znakova koji čine okolinu. Toj varijabli se može pristupati izravno ili korištenjem funkcija *getenv* i *putenv*.

```
char *getenv(char *name);
```

name pokazuje na niz znakova s *imenom* varijable okoline kojoj treba pristupiti. Rezultat je kazaljka na *vrijednost* te varijable u nizu oblika "*ime=vrijednost*" ili `NULL` ako varijabla nije nađena. Npr. ako u okolini postoji "*nadimak=pero*", tada će getenv("*nadimak*") vratiti pokazivač na "*pero*".

```
int putenv(char *string);
```

string pokazuje na niz znakova oblika "*ime=vrijednost*". *putenv* ga uključuje u okolinu umjesto postojećeg niza koji počinje istim *imenom* ili ga dodaje u okolinu. Rezultat je različit od 0 samo ako *putenv* nije dobio potrebnu memoriju za proširenje okoline.

putenv mijenja okolinu na koju pokazuje *environ* i kojoj se pristupa pomoću *getenv*. Međutim, pri tome se ne mijenja *envp* koji je predan funkciji *main*. Niz znakova na koji pokazuje kazaljka *string* postaje dio okoline.

POKRETANJE PROGRAMA (sustavski pozivi *exec*...)

Sustavski pozivi *exec* (u svim oblicima) inicijaliziraju proces novim programom. Jedino pomoću njih se izvršavaju programi u UNIX-u. Postoji šest poziva koji se uglavnom razlikuju po načinu prijenosa parametara (vidi `man exec`):

```
int execl(char *path, char *arg0, char *arg1,....., char *argn,
char *null)

int execv(char *path, char *argv[])

int execle(char *path, char *arg0,...., char *argn, char *null,
char *envp[])

int execve(char *path, char *argv[], char *envp[])

int execlp(char *file, char *arg0,...., char *argn, char *null)

int execvp(char *file, char *argv[])
```

Pozivom neke verzije poziva *exec* izvršava se navedeni program od početka, tj. pozivom funkcije *main*. Ako je poziv uspio, iz njega nema povratka. U slučaju greške rezultat je -1.

Argument *path* mora sadržavati put do datoteke sa izvršnom verzijom programa ili tekstom koji se može interpretirati (počinje sa `#!`) nekim drugim programom, najčešće ljuskom. Kod *execlp* i *execvp*, dovoljno je da argument *file* bude samo ime takve datoteke, a ona se traži u direktorijima koji su navedeni kao vrijednost varijable okoline "`PATH`".

execl, *execle* i *execlp* imaju varijabilan broj argumenata. Prvi argument *arg0* uvijek mora biti ime izvršne verzije prorama, a `NULL` je oznaka kraja argumenata. Od tih argumenata se kreira *argv* koji se predaje funkciji *main* novog programa.

Kod *execv*, *execve* i *execvp* predaje se izravno *argv*. Po dogovoru, i on mora imati barem jednu kazaljku koja pokazuje na niz znakova s imenom programa. Ostale pokazuju na argumente programa. Posljednja kazaljka mora biti `NULL` kako bi se znalo gdje je kraj i moglo izračunati *argc*.

envp u *execle* i *execve* je niz kazaljki na nizove znakova koji čine okolinu. Posljednja kazaljka mora biti `NULL`. Kod ostalih poziva, novi program dobiva postojeću okolinu (*environ*).

Otvoreni opisnici datoteka ostaju otvoreni kroz poziv *execl*. Ako to nije potrebno, treba ih zatvoriti prije nekog od ovih poziva. Kao i kod sustavskog poziva *fork*,

većina sustavskih atributa ostaje nepromjenjena.

Primjer upotrebe *exec* i *fork*

Obično *exec* služi za inicijalizaciju procesa djeteta kreiranog sustavskim pozivom *fork*. Slijedeći primjer pokazuje kako se *fork* i *exec* obično pozivaju:

```
switch (fork()) {

      case -1:
            printf("Ne mogu kreirati novi proces\n");
            break;
      case 0:
            execl("./ime","ime",NULL);
            exit(1);
      default:
            wait(NULL);
}
```

Ako *fork* ne uspije, rezultat je -1. Novi proces nije kreiran i dovoljno je ispisati odgovarajuću poruku ili pokušati ponovo. Ako je rezultat 0, nalazimo se u procesu djetetu i inicijaliziramo ga s programom `ime` bez dodatnih argumenata. Normalno nema povratka iz *execl*, ali ako on ne uspije, dijete ipak treba završiti sa *exit*. U slučaju nekog drugog rezultata poziva *fork*, radi se o nastavku procesa roditelja koji treba pričekati da dijete završi.

UPUTE za rad s naredbama ljuske operacijskog sustava za oslobađanje zauzetih računalnih resursa (zajedničke memorije, semafora i redova poruka) ukoliko dođe do nepredvidivog (!?) prekida izvođenja programa koji ih zauzima:

Naredba *ipcs*

Ova naredba daje informacije o sredstvima koja sudjeluju u komunikaciji među procesima. Bez opcija ispisuje informacije o postojećim redovima poruka, zajedničkoj memoriji i skupovima semafora.

Poziva se sa: `ipcs [ opcije ]`.

Opcije:

-q ispisuje informacije o aktivnim redovima poruka
-m ispisuje informacije o aktivnim segmentima zajedničke memorije
-s ispisuje informacije o aktivnim semaforima

Ako niti jedna od ovih opcija nije specificirana, tada se ispis može kontrolirati slijedećim opcijama:

-b ispisuje najveću dozvoljenu veličinu informacije (na primjer, najveći dozvoljeni broj bajtova u redu poruka)
-c ispisuje ime korisnika i njegove grupe
-o ispisuje broj poruka u redu i ukupan broj bajtova u redu poruka, odnosno broj procesa priključenih zajedničkoj memoriji
-p ispisuje identifikacijski broj procesa (koji je zadnji poslao poruku, priključio zajedničku memoriju i slično)
-t ispisuje informacije o vremenu koje ima nekakve veze sa semaforima, redovima poruka ili zajedničkom memorijom
-a upotrijebiti sve opcije

Stanje se može promjeniti dok se izvršava ova naredba. pa je slika koju daje samo približna.

Naredba *ipcrm*

Ova naredba uklanja red poruka, skup semafora ili oslobađa zajedničku memoriju. U stvari uklanjaju se identifikacijski brojevi. Poziva se sa: `ipcrm [ opcije ]`.

Opcije:

-q *msqid* uklanja identifikator reda poruka *msqid* iz sistema
-m *shmid* uklanja identifikator zajedničke memorije *shmid* iz sistema
-s *semid* uklanja identifikator semafora *semid*
-Q *msgkey* uklanja identifikator reda poruka koji je kreiran s ključem *msgkey*
-M *shmkey* uklanja identifikator zajedničke memorije zauzete s ključem *shmkey*
-S *semkey* uklanja identifikator semafora kreiranog s ključem *semkey*

Primjer rada s redovima poruka: kirk.c spock.c

Pripazite! U navedenim primjerima ključ koji se koristi prilikom dobavljanja reda poruka je postavljen na 12345. Ukoliko više studenata odjednom pokreće primjer, doći do greške. Naime, red poruka je već stvoren red s pravima pristupa 0600 i nitko drugi nema pravo slati ili čitati u taj red poruka! Stogo, promijenite ključ u primjerice identifikator korisnika - UID, kojeg možete dobaviti funkcijom getuid().